QUATRIÈME PARTIE

LES AUTRES PARADIGMES DE BASES DE DONNÉES : LES BASES NOSQL

Bibliographie

A First Course in DATABASE SYSTEMS
3rd edition
J. D. Ullman, J. Widom
Pearson, Prentice Hall, 2008

NoSQL Distilled
A Brief Guide to the Emerging World of Polyglot Persistence
P. J. Sadalage, M. Fowler
Addison-Wesley, 2013

Documentation en ligne PostgreSQL
http://www.postgresql.org/docs/

Origine du mot

- **NoSQL** : reconnu maintenant comme *Not Only SQL*.

Mais, que signifie vraiment « Not Only » ???

Origine du mot

- **NoSQL** : reconnu maintenant comme *Not Only SQL*.

Mais, que signifie vraiment « Not Only » ??? Pas grand chose... Pas formellement défini

- **NoSQL** → Open-source, distributed, non-relational databases

Origine du mot

- **NoSQL** : reconnu maintenant comme *Not Only SQL*.

Mais, que signifie vraiment « Not Only » ??? Pas grand chose... Pas formellement défini

- **NoSQL** → Open-source, distributed, non-relational databases

Historique

Le terme « NoSQL » dans sa signification actuelle vient du nom d'un séminaire (pendant une conférence sur Hadoop) qui a eu lieu en 2009 sur les nouveaux types de bases de données qui n'utilisent pas SQL. Il fallait un nom qui fasse un bon hashtag sur Twitter : court, facile à mémoriser, et non populaire sur Google pour espérer pouvoir remonter dans les premiers résultats.

Les systèmes du type NoSQL sont en fait utilisés depuis longtemps :
LDAP, stockage des cookies des navigateurs, ...

Rappels sur les bases de données relationnelles

Les principales opérations (LDD et LMD, SQL–92) :

- Interrogation (*select*)
- Insertion (*insert*)
- Mise à jour (*update*)
- Suppression (*delete)*

Rappels sur les bases de données relationnelles

Les principales opérations (LDD et LMD, SQL–92) :

- Interrogation (*select*)
- Insertion (*insert*)
- Mise à jour (*update*)
- Suppression (*delete)*

sont regroupées dans des **transactions**.

Une **transaction** est un regroupement atomique d'un ensemble d'opérations.

Rappels sur les bases de données relationnelles

Propriétés ACID des transactions :

- **A**tomicité
- **C**ohérence (Consistency)
- **I**solation
- **D**urabilité

Rappels sur les bases de données relationnelles

- **Atomicité**

  Toute transaction doit s'exécuter entièrement sans erreur ou pas du tout (ne doit alors laisser aucune trace de son existence).

Rappels sur les bases de données relationnelles

- **Atomicité**

  Toute transaction doit s'exécuter entièrement sans erreur ou pas du tout (ne doit alors laisser aucune trace de son existence).

- **Cohérence (Consistency)**

  La base de données doit toujours être dans un état cohérent entre deux transactions (même si une transaction ne se termine pas correctement).

Rappels sur les bases de données relationnelles

- **Isolation**

  Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions.

Rappels sur les bases de données relationnelles

- **Isolation**

  Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions.

- **Durabilité**

  Quand la transaction est terminée, la base de données doit être dans un état stable et durable dans le temps (persistance).

Rappels sur les bases de données relationnelles

Problèmes à éviter lors du déroulement des transactions

- *Lectures impropres*
  Une transaction lit des données écrites par une autre transaction non validée.

- *Lectures non reproductibles*
  Une transaction relit des données (donc déjà lues précédemment) mais les données ont été modifiées (par une autre transaction validée entre les deux lectures).

- *Lectures fantômes*
  Une même requête exécutée dans deux transactions ne renvoie pas le même résultat.

Exemples

Une transaction effectue un transfert de 1000€ du compte A vers le compte B :

1. Lire(A)
2. A := A − 1000
3. Ecrire(A)
4. Lire(B)
5. B := B + 1000
6. Ecrire(B)

Exemples

Une transaction effectue un transfert de 1000€ du compte A vers le compte B :

1. Lire(A)
2. A := A − 1000
3. Ecrire(A)
4. Lire(B)
5. B := B + 1000
6. Ecrire(B)

- **Atomicité** : Si la transaction “échoue” après l’étape 3, alors le système doit s’assurer que les modifications de A ne soient pas persistantes

Rappels sur les bases de données relationnelles

Exemples

Une transaction effectue un transfert de 1000€ du compte A vers le compte B :

1. Lire(A)
2. A := A − 1000
3. Ecrire(A)
4. Lire(B)
5. B := B + 1000
6. Ecrire(B)

- **Atomicité** : Si la transaction “échoue” après l’étape 3, alors le système doit s’assurer que les modifications de A ne soient pas persistantes
- **Cohérence** : la base est cohérente si la somme A+B ne change pas suite à l’exécution de la transaction.

Rappels sur les bases de données relationnelles

Exemples

Une transaction effectue un transfert de 1000€ du compte A vers le compte B :

1. Lire(A)
2. A := A − 1000
3. Ecrire(A)
4. Lire(B)
5. B := B + 1000
6. Ecrire(B)

- **Atomicité** : Si la transaction “échoue” après l’étape 3, alors le système doit s’assurer que les modifications de A ne soient pas persistantes
- **Cohérence** : la base est cohérente si la somme A+B ne change pas suite à l’exécution de la transaction.
- **Durabilité :** Une fois l’utilisateur est informé que la transaction est validée, il n’a plus à s’inquiéter du sort de son transfert

Limites des bases de données relationnelles

Relation

Contient un ensemble de *n*-uplets (les lignes) qui contiennent des attributs (les colonnes). Le domaine de chaque attribut est fixé et doit être composé de valeurs atomiques simples, *i.e.* non décomposables (1FN).

Une requête SQL de type *select* retourne toujours une relation.

Limites des bases de données relationnelles

Relation

Contient un ensemble de *n*-uplets (les lignes) qui contiennent des attributs (les colonnes). Le domaine de chaque attribut est fixé et doit être composé de valeurs atomiques simples, *i.e.* non décomposables (1FN).
Une requête SQL de type *select* retourne toujours une relation.

Comment stocker des structures plus complexes ?

Possible dans la plupart des langages de programmation (notamment à objets) mais pas avec le modèle relationnel afin de respecter la normalisation et la définition d'une relation.
Un changement de modèle et une traduction des données sont donc nécessaires.

Limites des bases de données relationnelles

Relation
Contient un ensemble de *n*-uplets (les lignes) qui contiennent des attributs (les colonnes). Le domaine de chaque attribut est fixé et doit être composé de valeurs atomiques simples, *i.e.* non décomposables (1FN).
Une requête SQL de type *select* retourne toujours une relation.

Comment stocker des structures plus complexes ?
Possible dans la plupart des langages de programmation (notamment à objets) mais pas avec le modèle relationnel afin de respecter la normalisation et la définition d'une relation.
Un changement de modèle et une traduction des données sont donc nécessaires.

Impedance mismatch

Pour la petite histoire : tentative de solutions avec les BD objets dans les années 1990.

Limites des bases de données relationnelles

- Une BD relationnelle est une couche spécifique pour accéder aux données (architecture à trois couches (niveaux, tiers) ; 3-tier en anglais).

Limites des bases de données relationnelles

- Une BD relationnelle est une couche spécifique pour accéder aux données (architecture à trois couches (niveaux, tiers) ; 3-tier en anglais).
- De nos jours, intégration de la base de données dans l'application pour simplifier l'architecture du système et accéder plus rapidement aux données (pas de serveur dédié).

Limites des bases de données relationnelles

- Une BD relationnelle est une couche spécifique pour accéder aux données (architecture à trois couches (niveaux, tiers) ; 3-tier en anglais).
- De nos jours, intégration de la base de données dans l'application pour simplifier l'architecture du système et accéder plus rapidement aux données (pas de serveur dédié).
- Les BD relationnelles sont conçues pour fonctionner sur un serveur unique avec des volumes de données modérés (maintenance des index, maintien de la cohérence).

Limites des bases de données relationnelles

- Une BD relationnelle est une couche spécifique pour accéder aux données (architecture à trois couches (niveaux, tiers) ; 3-tier en anglais).
- De nos jours, intégration de la base de données dans l'application pour simplifier l'architecture du système et accéder plus rapidement aux données (pas de serveur dédié).
- Les BD relationnelles sont conçues pour fonctionner sur un serveur unique avec des volumes de données modérés (maintenance des index, maintien de la cohérence).
- Contraintes ACID impossibles à respecter sur un cluster (et *a fortiori* sur Internet) pour conserver un fonctionnement correct (haute disponibilité, rapidité d'exécution et cohérence permanente de la base, partition accidentelle du cluster).

Les systèmes NoSQL

- *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable* (http://nosql-database.org/).

Les systèmes NoSQL

- *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable* (http://nosql-database.org/).
- Pas un Système de Gestion de Bases de Données mais plus un système de stockage de données.

Les systèmes NoSQL

- *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable* (http://nosql-database.org/).
- Pas un Système de Gestion de Bases de Données mais plus un système de stockage de données.
- Pas de modèle pré-défini, pas de contrainte stricte comme en relationnel, tout doit rester simple pour aller vite. L'ajout d'un serveur doit améliorer les performances.

Les systèmes NoSQL

- *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable* (http://nosql-database.org/).
- Pas un Système de Gestion de Bases de Données mais plus un système de stockage de données.
- Pas de modèle pré-défini, pas de contrainte stricte comme en relationnel, tout doit rester simple pour aller vite. L'ajout d'un serveur doit améliorer les performances.
- Maintien de la cohérence simple. Exemple du QUORUM : Soit un nombre de serveurs *n*. Dès que *w* serveurs sur les *n* ont confirmé la bonne exécution d'un ordre d'écriture, l'écriture est considérée comme valide sur l'ensemble du cluster. Les autres serveurs finiront par se mettre à jour. En relationnel, il faudrait attendre que tous les serveurs aient confirmé.

Les systèmes NoSQL : CAP

Théorème CAP (proposé par Brewer, 2000 et démontré par Gilbert et Lynch 2002)

- Dans un environnement distribué, il n'est pas possible de respecter simultanément :
  - **C**onsistency,
  - **A**vailability
  - **P**artition Tolerance

Les systèmes NoSQL : CAP

Théorème CAP (proposé par Brewer, 2000 et démontré par Gilbert et Lynch 2002)

- Dans un environnement distribué, il n'est pas possible de respecter simultanément :
  - **C**onsistency,
  - **A**vailability (tout serveur accessible permet de faire des lectures/écritures),
  - **P**artition Tolerance (si le cluster est accidentellement partionné, le système doit continuer à fonctionner)

Les systèmes NoSQL : CAP

Théorème CAP (proposé par Brewer, 2000 et démontré par Gilbert et Lynch 2002)

- Dans un environnement distribué, il n'est pas possible de respecter simultanément :
  - **C**onsistency,
  - **A**vailability (tout serveur accessible permet de faire des lectures/écritures),
  - **P**artition Tolerance (si le cluster est accidentellement partionné, le système doit continuer à fonctionner)
- On peut, en revanche, respecter deux contraintes.
- BD relationnelles : CA

Les systèmes NoSQL

À l'opposé de ACID, les systèmes NoSQL sont parfois présentés comme BASE :

Basically Available, Soft state, Eventually consistent

Les systèmes NoSQL

À l'opposé de ACID, les systèmes NoSQL sont parfois présentés comme BASE :

Basically Available, Soft state, Eventually consistent

- **Eventual consistency :** la cohérence du système est atteinte au bout d'un certain temps.

Les systèmes NoSQL

À l'opposé de ACID, les systèmes NoSQL sont parfois présentés comme BASE :

Basically Available, Soft state, Eventually consistent

- **Eventual consistency :** la cohérence du système est atteinte au bout d'un certain temps.
- **Soft state :** l'état du système peut donc changer même sans action de l'utilisateur pour atteindre un état global cohérent. Il faut donc que le système soit capable de fonctionner dans des états transitoires.

Les systèmes NoSQL

À l'opposé de ACID, les systèmes NoSQL sont parfois présentés comme BASE :

Basically Available, Soft state, Eventually consistent

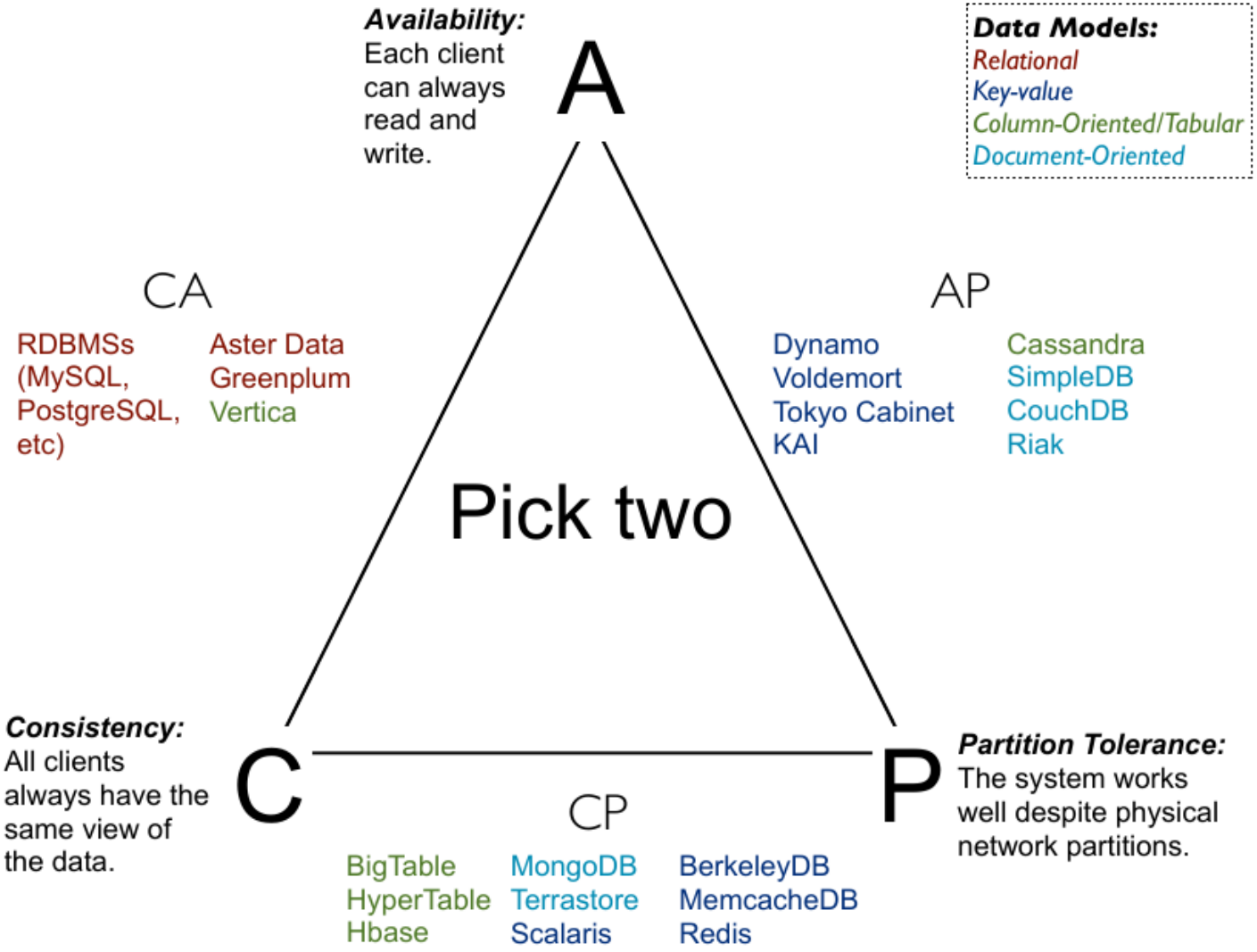
- **Eventual consistency :** la cohérence du système est atteinte au bout d'un certain temps.
- **Soft state :** l'état du système peut donc changer même sans action de l'utilisateur pour atteindre un état global cohérent. Il faut donc que le système soit capable de fonctionner dans des états transitoires.
- **Basically available** : le système reste donc globalement opérationnel

Les systèmes NoSQL : choix

4 grands types de systèmes

- Orienté clé-valeurs
- Orienté documents
- Orienté colonnes
- Orienté graphes

Les systèmes NoSQL : choix



Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.

Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.

- Système le plus simple : lecture, écriture et suppression. Ne fait que stocker des données en rapport avec une clé.

Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.

- Système le plus simple : lecture, écriture et suppression. Ne fait que stocker des données en rapport avec une clé.

- Implique pas de relation entre les données. Le système ne connaît pas les données ou ce qu'elles représentent (sémantique associée).

Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.

- Système le plus simple : lecture, écriture et suppression. Ne fait que stocker des données en rapport avec une clé.

- Implique pas de relation entre les données. Le système ne connaît pas les données ou ce qu'elles représentent (sémantique associée).

- Interrogation uniquement par la clé. On récupère une donnée (blob notamment) associée à une clé.

Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.
- Système le plus simple : lecture, écriture et suppression. Ne fait que stocker des données en rapport avec une clé.
- Implique pas de relation entre les données. Le système ne connaît pas les données ou ce qu'elles représentent (sémantique associée).
- Interrogation uniquement par la clé. On récupère une donnée (blob notamment) associée à une clé.
- **Exemples d'usages** : stockage d'informations de session, profil utilisateurs, préférences, paniers d'achats

Les systèmes NoSQL : Documents

- Basé sur un système clé-valeur
- Mais la valeur est une structure que le système connaît et peut donc parcourir.

Les systèmes NoSQL : Documents

- Basé sur un système clé-valeur
- Mais la valeur est une structure que le système connaît et peut donc parcourir.
- Pour la plupart des systèmes pas d'opérations croisées entres plusieurs documents.

Les systèmes NoSQL : Documents

- Basé sur un système clé-valeur
- Mais la valeur est une structure que le système connaît et peut donc parcourir.
- Pour la plupart des systèmes pas d'opérations croisées entres plusieurs documents.
- **Exemples d'usages** : stockage d’événements (event logging), CMS et dérivés, applications e-commerce.

Les systèmes NoSQL : colonnes

- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

Les systèmes NoSQL : colonnes

- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

- Les colonnes sont stockées sous une forme clé-valeur. On peut ajouter une colonne dans un enregistrement aussi facilement que l'ajout d'un n-uplet en relationnel.

Les systèmes NoSQL : colonnes

- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

- Les colonnes sont stockées sous une forme clé-valeur. On peut ajouter une colonne dans un enregistrement aussi facilement que l'ajout d'un n-uplet en relationnel.

	A	B	C	D	E
1	foo	bar	hello		
2		Tom			
3			java	scala	cobol

Organisation d'une table dans une BDD relationnelle

http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra/

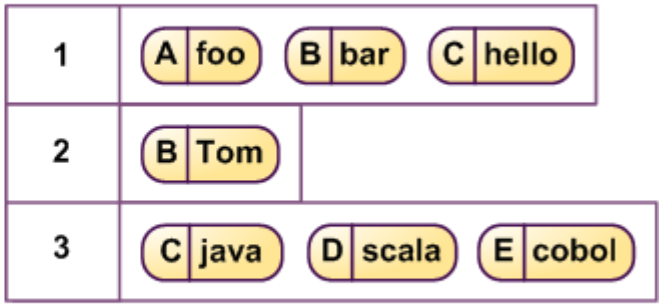
Les systèmes NoSQL : colonnes

- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

- Les colonnes sont stockées sous une forme clé-valeur. On peut ajouter une colonne dans un enregistrement aussi facilement que l'ajout d'un n-uplet en relationnel.

	A	B	C	D	E
1	foo	bar	hello		
2		Tom			
3			java	scala	cobol

Organisation d'une table dans une BDD relationnelle



Organisation d'une table dans une BDD orientée colonnes

http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra/

Les systèmes NoSQL : colonnes

- Coût de stockage d'une valeur nulle :

Les systèmes NoSQL : colonnes

- Coût de stockage d'une valeur nulle : 0
- Stockage de plusieurs millions de colonnes

Les systèmes NoSQL : colonnes

- Coût de stockage d'une valeur nulle : 0
- Stockage de plusieurs millions de colonnes
- **Exemples d'usages** : stockage d’événements (event logging), CRM (Customer Relationship Management), CMS (Content Management System), compteurs (comptage et trie des visiteurs passant sur une page d'un site)

Les systèmes NoSQL : Graphes

- On ne stocke plus un simple ensemble de données mais des relations.

Les systèmes NoSQL : Graphes

- On ne stocke plus un simple ensemble de données mais des relations.
- Stockage d'entités et de relations entre les entités. On peut ajouter des propriétés aux relations et aux entités.

Les systèmes NoSQL : Graphes

- On ne stocke plus un simple ensemble de données mais des relations.
- Stockage d'entités et de relations entre les entités. On peut ajouter des propriétés aux relations et aux entités.
- Cohérence forte des données. Pas d'arête pendante.
- Difficile de monter en charge (ajout de serveurs). Le stockage d'un graphe sur plusieurs serveurs est un problème difficile (problème de performance même pour un simple calcul de chemin si le graphe est réparti sur plusieurs serveurs).

Les systèmes NoSQL : Graphes

- On ne stocke plus un simple ensemble de données mais des relations.
- Stockage d'entités et de relations entre les entités. On peut ajouter des propriétés aux relations et aux entités.
- Cohérence forte des données. Pas d'arête pendante.
- Difficile de monter en charge (ajout de serveurs). Le stockage d'un graphe sur plusieurs serveurs est un problème difficile (problème de performance même pour un simple calcul de chemin si le graphe est réparti sur plusieurs serveurs).

- **Exemples d'usages** : réseaux sociaux, système de recommandations, services géolocalisés

Conclusion

Conclusion

- Les SGBDR sont maintenant une option parmi un panel de plus en plus large.

- Il faut apprendre à utiliser la base de données la plus adaptée aux données à gérer quitte à en utiliser plusieurs.

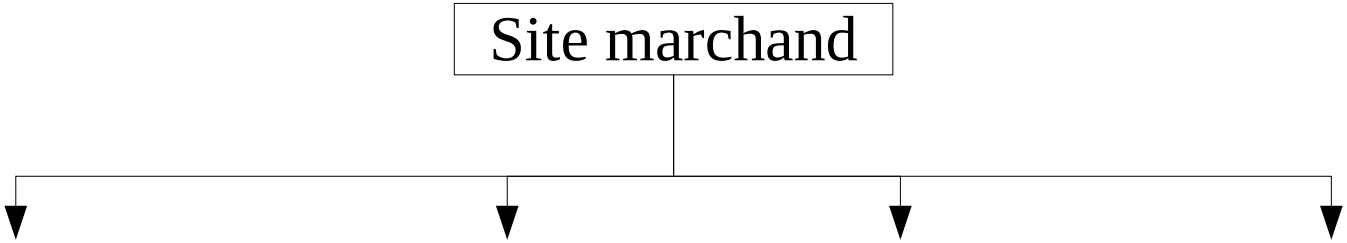
Conclusion

- Les SGBDR sont maintenant une option parmi un panel de plus en plus large.

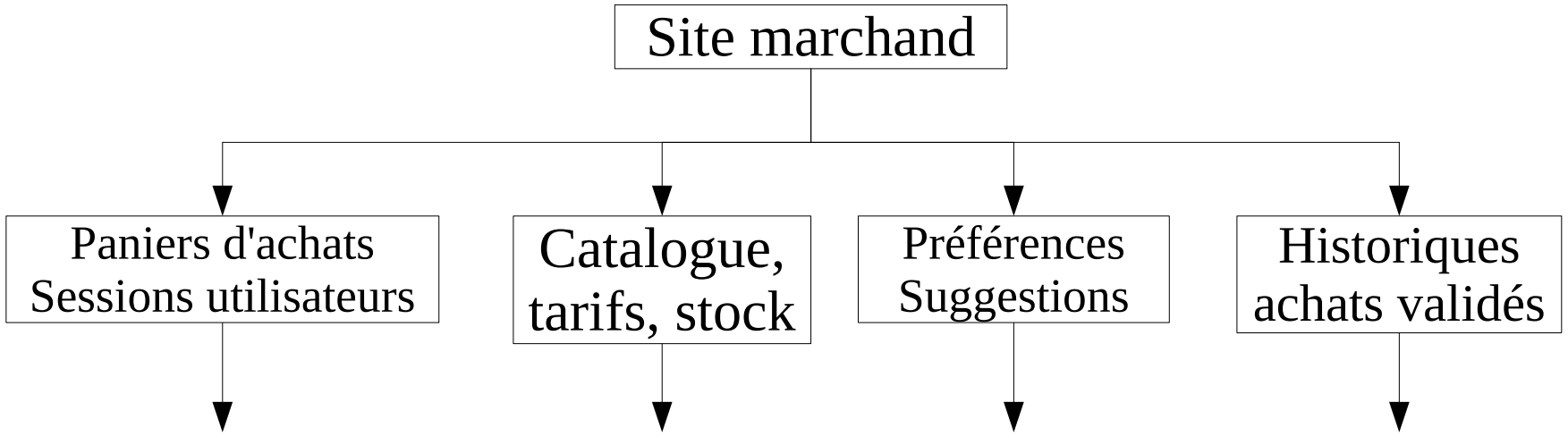
- Il faut apprendre à utiliser la base de données la plus adaptée aux données à gérer quitte à en utiliser plusieurs.

- Déplacement de la base de données vers les développeurs d'applications au lieu de la centralisation par la DSI

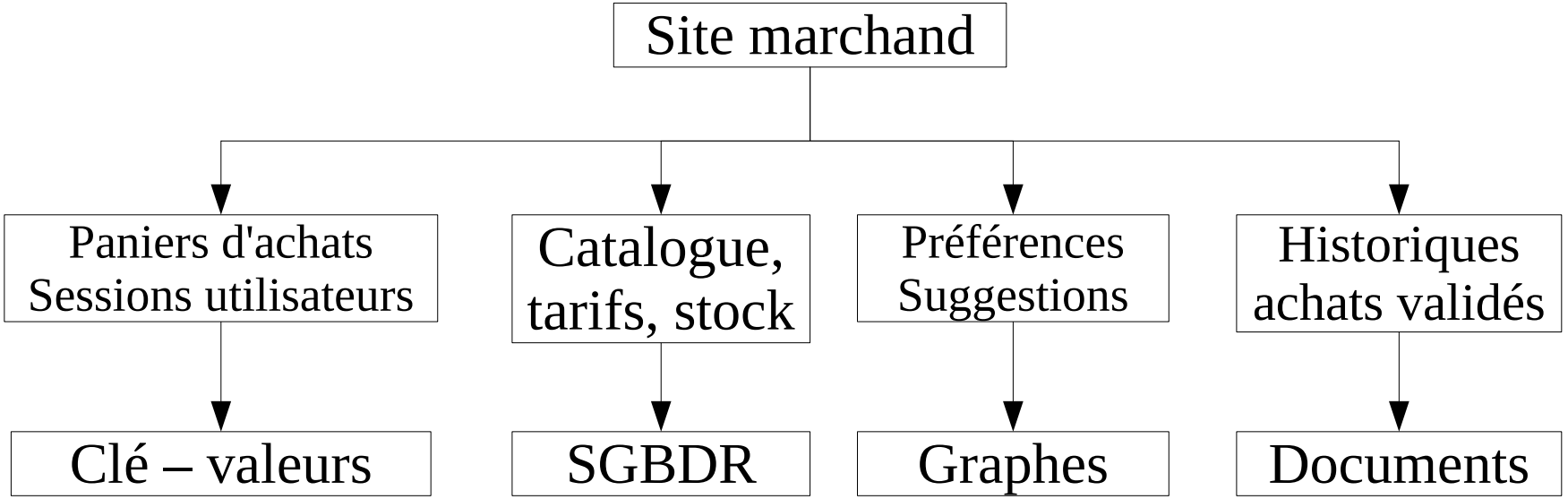
Conclusion



Conclusion


Site marchand
Paniers d'achats
Sessions utilisateurs
Catalogue,
tarifs, stock
Préférences
Suggestions
Historiques
achats validés

Conclusion


Site marchand
Paniers d'achats
Sessions utilisateurs
Catalogue,
tarifs, stock
Préférences
Suggestions
Historiques
achats validés
Clé – valeurs
SGBDR
Graphes
Documents

Conclusion

• Les systèmes NoSQL ne remplaceront jamais les bases de données relationnelles. Il faut les voir comme une alternative après un quasi monopole du relationnel de plusieurs décennies (*One size fits all*).

• Les systèmes de gestion de données relationnelles sont maintenant une option parmi d'autres. Leur généricité leur permet d'être utilisés dans de nombreux cas mais les rends peu performants et complexes sur certains types de données (masse de données, attributs trop nombreux, …).

• Il faut savoir utiliser la solution la plus adaptée et pourquoi pas en utiliser plusieurs.

• Pas de schéma figé comme en relationnel, moins de contraintes. On déporte sur l'application la gestion du modèle de données et sa connaissance.

• Changement drastique des méthodes de développements notamment par l'absence de cohérence immédiate pour la tolérance au partitionnement et les évolutions du schéma.